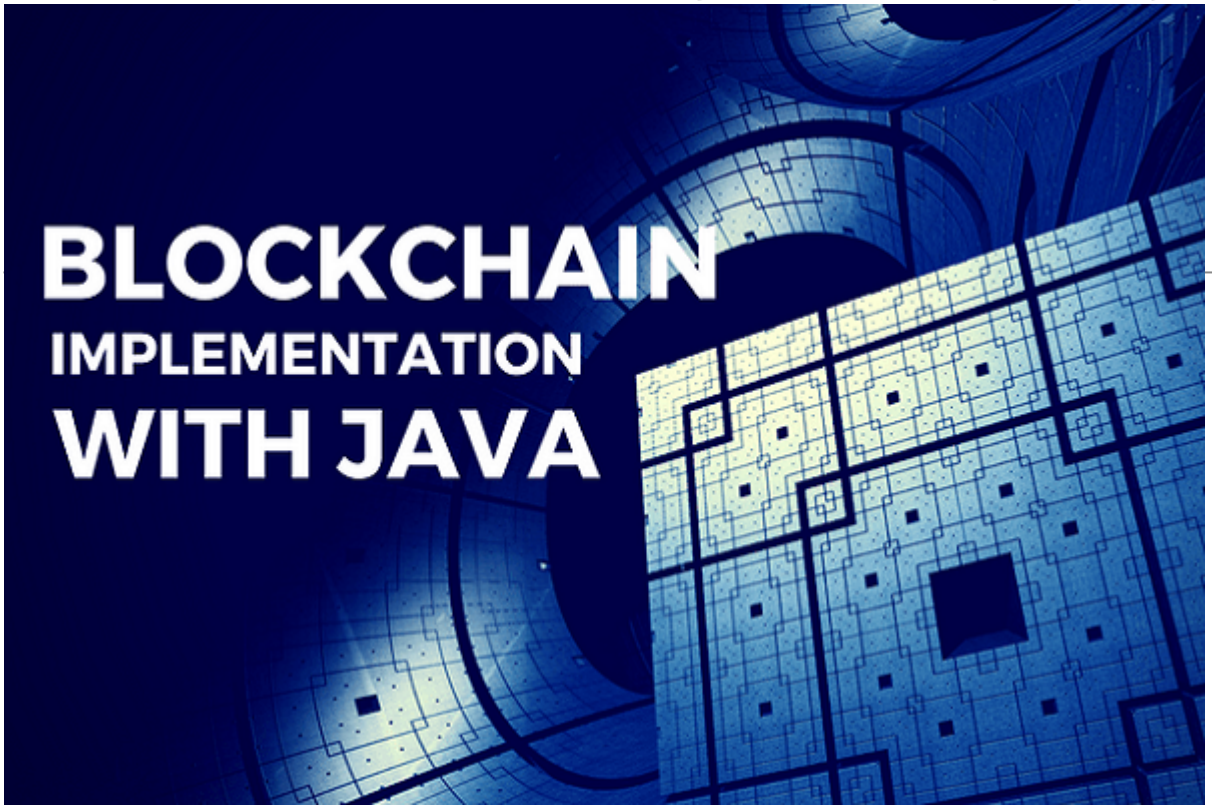
زنجیره بلوکی به‌عنوان یک بستر قدرتمند

پیاده‌سازی زنجیره بلوکی با کدهای جاوا



با وجود این‌که آینده ارزرمزها در هاله‌ای از ابهام قرار دارد، زنجیره بلوکی (همان فناوری که برای فعالیت بیت‌کوین از آن استفاده می‌شود) به‌عنوان یک بستر قدرتمند در حال کار است. محدوده کاربرد زنجیره بلوکی تقریبا بی‌انتها است و به یقین این توانایی بالقوه را دارد که اتوماسیون اداری را متحول کند. درباره چگونگی کارکرد زنجیره بلوکی منابع اطلاعاتی فراوانی وجود دارد که برای درک بهتر نحوه عملکرد این فناوری می‌توانید به آن مراجعه کنید.
زمانی‌که به‌عنوان یک توسعه‌دهنده کدها را بررسی کنید، درک و شناخت بهتری نسبت به زمانی که تنها مقاله‌های فنی را مطالعه می‌کنید خواهید داشت.
این مطلب یکی از مقالات پرونده ویژه«بلاک‌چین، کی، کِی، کجا، چگونه، چرا» شماره 213 ماهنامه شبکه است. علاقه‌مندان می‌توانند کل این پرونده ویژه را از روی سایت شبکه دانلود کنند.

زنجیره بلوکی در چند کلمه

در ابتدا نگاهی اجمالی به فناوری زنجیره بلوکی خواهیم داشت. یک بلوک شامل یکسری اطلاعات هدر و یک مجموعه یا بلوک از تراکنش‌ها از هر نوع داده‌ای است. این زنجیره با اولین بلوک آغاز می‌شود. زمانی‌که تراکنش‌ها اضافه یا الصاق می‌شوند، بلوک‌های جدید بر اساس تعداد تراکنش‌هایی که می‌تواند درون یک بلوک ذخیره شود، ایجاد می‌شوند. هنگام تکمیل ظرفیت یک بلوک، بلوکی جدید از تراکنش‌ها ساخته می‌شود و این بلوک جدید به بلوک قبلی متصل شده و در اصطلاح **زنجیره بلوکی** پدید می‌آید.

پابرجایی

زنجیره‌های بلوکی پابرجا و تغییرناپذیر هستند، زیرا در آن از یک سیستم رمزنگاری SHA-256 برای محاسبات تراکنش‌ها استفاده می‌شود. محتوای یک بلوک رمزگذاری شده و با یک شناسه واحد شناسایی می‌شود. علاوه بر این، اطلاعات رمزنگاری بلوک‌های قبلی در هدر این بلوک ذخیره و رمزگذاری می‌شود. به همین دلیل تلاش برای رخنه به یک بلوک از زنجیره بلوکی (دست‌کم با قدرت محاسبات امروزی) عملا غیرممکن خواهد بود. در زیر بخشی از یک کلاس جاوا را مشاهده می‌کنید که خصوصیات یک بلوک را نمایش می‌دهد.

```
...
public class Block&lt;T extends Tx&gt; {
public long timeStamp;
private int index;
private List&lt;T&gt; transactions = new ArrayList&lt;T&gt;();
private String hash;
```

```
private String previousHash;
private String merkleRoot;
private String nonce = “0000”;
// caches Transaction SHA256 hashes
    public Map&lt;String,T&gt; map = new HashMap&lt;String,T&gt;();
...
```

توجه داشته باشید، نوع متغیر تزریقی از نوع Tx است که اجازه می‌دهد اطلاعات مبادله شده متفاوت باشد. همچنین خصوصیت PreviousHash به هش بلاک قبلی ارجاع داده خواهد شد.

Block Hash

هر بلوک می‌تواند یک Block Hash را محاسبه کند. این شامل یک هش از تمام خصوصیات این بلوک است که با یکدیگر مرتبط می‌شوند و شامل هش بلوک قبلی و یک هش SHA-256 محاسبه‌شده از آن است.
در اینجا متد تعریف‌شده در کلاس Block.java را مشاهده می‌کنید که این هش را محاسبه می‌کند.

```
...
public void computeHash() {
     Gson parser = new Gson(); // probably should cache this instance
     String serializedData = parser.toJson(transactions);
     setHash(SHA256.generateHash(timeStamp + index + merkleRoot + serializedData + nonce
+ previousHash));
     }
...
```

تراکنش‌های بلوک به شکل سریالی درون یک‌رشته JSON نگهداری می‌شود و بنابراین می‌توان آن را قبل از رمزگذاری به خصوصیات بلوک الصاق کرد.

مطلب پیشنهادی

جاوا اسکریپت بی‌حد و مرز
آینده توسعه برنامه‌های کاربردی در دستان جاوااسکریپت است

زنجیره

زنجیره بلوکی با پذیرش تراکنش‌ها بلوک‌ها را مدیریت می‌کند. هنگامی‌که ظرفیت تعریف‌شده برای هر بلوک به اتمام می‌رسد، یک بلوک دیگر ساخته می‌شود. در اینجا نحوه پیاده‌سازی یک SimpleBlockChain.java را مشاهده می‌کنید:

```
...
...
public class SimpleBlockchain<T extends Tx> {
public static final int BLOCK_SIZE = 10;
public List<Block<T>> chain = new ArrayList<Block<T>>();
public SimpleBlockchain() {
// create genesis block
chain.add(newBlock());
}
...
```

توجه داشته باشید خصوصیت Chain فهرستی از نوع بلوک‌ها را با یک نوع Tx نگهداری می‌کند. همچنین سازنده No arg یک بلوک مقدماتی را بعد از ساخته‌شدن این زنجیره ایجاد می‌کند. در زیر متد newBlock() را مشاهده می‌کنید:

```
...
public Block<T> newBlock() {
int count = chain.size();
String previousHash = “root”;
if (count > 0)
previousHash = blockChainHash();
Block<T> block = new Block<T>();
block.setTimeStamp(System.currentTimeMillis());
block.setIndex(count);
block.setPreviousHash(previousHash);
return block;
}
...
```

این متد بلوک جدید یک نمونه از بلوک جدید را ایجاد و مقادیر لازم را فراهم کرده و هش بلوک قبلی را تعیین می‌کند و در انتها بلوک را برمی‌گرداند.
می‌توان بلوک‌ها را قبل از اضافه شدن به زنجیره با مقایسه هش بلوک قبلی با آخرین بلوک (head) زنجیره اعتبار سنجی کرد. در زیر یک متد SimpleBlockchain.java را مشاهده می‌کنید که این کار را انجام می‌دهد:

```
....
public void addAndValidateBlock(Block<T> block) {
// compare previous block hash, add if valid
Block<T> current = block;
for (int i = chain.size() - 1; i >= 0; i--) {
Block<T> b = chain.get(i);
if (b.getHash().equals(current.getPreviousHash())) {
current = b;
} else {
throw new RuntimeException(“Block Invalid”);
}
}
this.chain.add(block);
}
...
```

تمام زنجیره بلوکی از طریق حلقه تکرار زنجیره اعتبار سنجی می‌شود تا این اطمینان حاصل شود که یک هش بلوک همچنان با هش بلوک قبل از خود سازگار است.
پیاده‌سازی متد SimpleBlockChain.java validate() به این صورت است:

```
...
public boolean validate() {
String previousHash = null;
for (Block<T> block : chain) {
String currentHash = block.getHash();
if (!currentHash.equals(previousHash)) {
return false;
```

```
}
previousHash = currentHash;
}
return true;
}
...
```

ملاحظه می‌کنید که با استفاده از چنین روشی دستیابی به داده‌های تراکنش یا هر خصوصیت دیگری بسیار دشوار خواهد بود و هر چه این زنجیره بزرگ‌تر می‌شود، دستیابی به آن نیز دشوارتر می‌شود و تا زمان حضور کامپیوترهای کوانتومی به دست آوردن آن با توان محاسباتی حال حاضر غیرممکن خواهد بود.

مطلب پیشنهادی

جاوااسکریپت در تعامل با ارزهای دیجیتالی
جاوااسکریپت در حال بلعیدن زنجیره بلوکی (بلاک‌چین) است

اضافه کردن تراکنش‌ها

یکی دیگر از جنبه‌های فنی فناوری زنجیره بلوکی قابلیت توزیع‌پذیری آن است. خاصیت الصاق‌پذیری آن کمک می‌کند تا **زنجیره‌های بلوکی** به‌راحتی به گروه‌های موجود در یک شبکه زنجیره بلوکی اضافه شوند. گره‌ها معمولا به شیوه نظیربه‌نظیر و با همان ساختار بیت‌کوین با یکدیگر ارتباط برقرار می‌کنند. سایر روش‌های دیگر پیاده‌سازی زنجیره بلوکی از ساختار غیرمتمرکز مثل استفاده از APIها از طریق HTTP استفاده می‌کند. اما این موضوعی است که باید در یک مقاله جداگانه به آن پرداخته شود.

ساختار داده Merkle Tree

تراکنش‌ها رمزگذاری شده هستند و به بلوک اضافه می‌شوند. یک ساختار داده Merkle Tree ساخته می‌شود تا یک رمزنگاری Merkle Root را محاسبه کند. هر بلوک ریشه درخت Merkle را ذخیره می‌کند. از این ساختار درختی برای معتبر ساختن تراکنش‌های بلوک استفاده می‌شود. اگر حتی یک بیت از اطلاعات در هر تراکنشی تغییر کند، Merkle Root نامعتبر خواهد شد.
در زیر متدی از کلاس Block.java را مشاهده می‌کنید که یک Merkle Tree را از فهرست تراکنش ایجاد می‌کند:

```
...
public List<String> merkleTree() {
ArrayList<String> tree = new ArrayList<>();
// Start by adding all the hashes of the transactions as leaves of the
// tree.
for (T t : transactions) {
tree.add(t.hash());
}
int levelOffset = 0; // Offset in the list where the currently processed
// level starts.
// Step through each level, stopping when we reach the root (levelSize
// == 1).
for (int levelSize = transactions.size(); levelSize > 1; levelSize = (levelSize + 1) / 2) {
// For each pair of nodes on that level:
for (int left = 0; left < levelSize; left += 2) {
// The right hand node can be the same as the left hand, in the
```

```
// case where we don’t have enough
// transactions.
int right = Math.min(left + 1, levelSize - 1);
String tleft = tree.get(levelOffset + left);
String tright = tree.get(levelOffset + right);
tree.add(SHA256.generateHash(tleft + tright));
}
// Move to the next level.
levelOffset += levelSize;
}
return tree;
}
...
```

از این متد برای محاسبه ریشه Merkle Tree برای یک بلوک استفاده می‌شود. این پروژه یک واحد Merkle Tree آزمایشی دارد که تلاش می‌کند یک تراکنش را به یک بلوک اضافه کند و تایید می‌کند Merkle Roots تغییر پیدا کرده است. در اینجا کد منبع این واحد Merkle Tree آزمایشی را مشاهده می‌کنید.

```
...
Test@
public void merkleTreeTest() {
// create chain, add transaction
SimpleBlockchain<Transaction> chain1 = new SimpleBlockchain<Transaction>();
chain1.add(new Transaction(“A”)).add(new Transaction(“B”)).add(new Transaction(“C”)).add(new
Transaction(“D”));
// get a block in chain
Block<Transaction> block = chain1.getHead();
System.out.println(“Merkle Hash tree :” + block.merkleTree());
// get a transaction from block
Transaction tx = block.getTransactions().get(0);
// see if block transactions are valid, they should be
block.transasctionsValid();
assertTrue(block.transasctionsValid());
// mutate the data of a transaction
tx.setValue(“Z”);
// block should no longer be valid, blocks MerkleRoot does not match computed merkle tree of
transactions
;(()assertFalse(block.transasctionsValid
{
...
```

این کد تاییدیه تراکنش‌ها و بعد تغییرات تراکنش داخل یک بلوک خارج از مکانیسم اجماع (برای مثال، زمانی که شخصی سعی می‌کند داده تراکنش را تغییر دهد) را شبیه‌سازی می‌کند. به یاد داشته باشید زنجیره‌های بلوکی را تنها می‌توان الصاق کرد و بعد از این‌که ساختار داده زنجیره بلوکی بین گره‌ها به اشتراک گذاشته شد، ساختار داده بلوک (از جمله Merkle Root) کدگذاری شده و به سایر بلوک‌ها متصل می‌شود. تمام گره‌ها می‌توانند بلوک‌های جدید را تایید کنند و بلوک‌های موجود را می‌توان به‌راحتی بررسی و تایید کرد. بنابراین یک ماینر دیگر نمی‌تواند یک بلوک جعلی را به زنجیره اضافه کند.

مطلب پیشنهادی

همه مصائب زنجیره بلوکی
زنجیره بلوکی چیست و چرا پیاده‌سازی آن فرآیندی پیچیده و دشوار است؟

mining

فرآیند ادغام تراکنش‌ها به یک بلوک و سپس اعتبارسنجی آن به‌وسیله اعضای زنجیره در دنیای بیت‌کوین به Mining معروف است.
گره‌های Mining منتظر اجرا شدن تراکنش‌ها توسط زنجیره بلوکی می‌مانند و یک پازل ساده ریاضی را به اجرا می‌گذارند. پروژه جاوای مثال ما یک کلاس Miner.java دارد که یک متد proofOfWork(Block block) را اجرا می‌کند:

```
private String proofOfWork(Block block) {
String nonceKey = block.getNonce();
long nonce = 0;
boolean nonceFound = false;
String nonceHash = “”;
Gson parser = new Gson();
String serializedData = parser.toJson(transactionPool);
String message = block.getTimeStamp() + block.getIndex() + block.getMerkleRoot() +
serializedData
+ block.getPreviousHash();
while (!nonceFound) {
nonceHash = SHA256.generateHash(message + nonce);
nonceFound = nonceHash.substring(0, nonceKey.length()).equals(nonceKey);
nonce++;
}
return nonceHash;
}
```

این الگوریتم به‌سادگی یک حلقه تکرار و یک هش SHA-256 ایجاد می‌کند تا این‌که شماره هش اصلی تولید شود. این فرآیند بسیار زمان‌بر است. به همین دلیل برای پردازش هر چه سریع‌تر چنین فرآیندی از پردازنده‌های گرافیکی خاص استفاده می‌شود.

تاریخ انتشار:

16 اردیبهشت 1398

نشانی منبع:
https://www.shabakeh-mag.com/cover-story/14856/%D9%BE%DB%8C%D8%A7%D8%AF%D9%87%E2%80%8C%D8%B3%D8%A7%D8%B2%DB%8C-%D8%B2%D9%86%D8%AC%D9%8A%D8%B1%D9%87-%D8%A8%D9%84%D9%88%DA%A9%DB%8C-%D8%A8%D8%A7-%DA%A9%D8%AF%D9%87%D8%A7%DB%8C-%D8%AC%D8%A7%D9%88%D8%A7